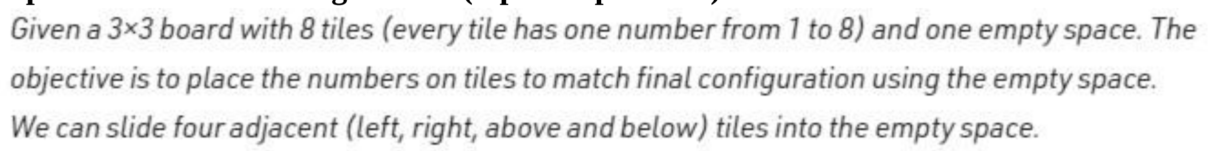
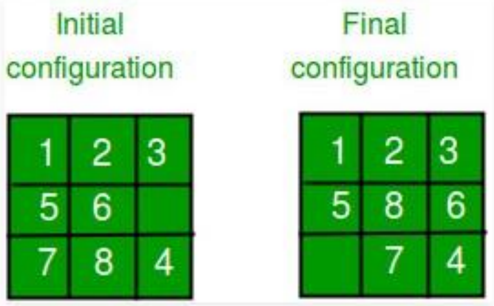
Introduction to Artificial Intelligence Lab

Experiment 5: Search Algorithms (8-puzzle problem)



Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

For example,



Initial configuration

1	2	3
5	6	
7	8	4

Final configuration

1	2	3
5	8	6
	7	4

Q1) Formulate 8-puzzle problem according artificial intelligence problem solving concept.
Q2) Solve the 8-puzzle problem using Breadth First Search (BFS) Algorithm (uninformed search algorithm).
Q3) Solve the 8-puzzle problem using Best First Search (Greedy Search) Algorithm (informed search algorithm).
Q4) Compare the algorithms in Q2 and Q3 according to following criteria.

a. Optimality
b. Completeness
c. Time Complexity
d. Space Complexity

Node.cs

```
namespace Lab5
{
    class Node
    {
        public List<Node> children = new List<Node>();
        public Node parent;
        public int[] puzzle = new int[9];
        public int x = 0;
        public int col = 3;

        public Node(int[] p)
        {
            SetPuzzle(p);
        }

        public void SetPuzzle(int[] p)
        {
            for (int i = 0; i < puzzle.Length; i++)
            {
                this.puzzle[i] = p[i];
        }
}

public void ExpandMove()
{
        for (int i = 0; i < puzzle.Length; i++)
        {
                if (puzzle[i] == 0)
                {
                        x = i;
                }
        }

        MoveToRight(puzzle, x);
        MoveToLeft(puzzle, x);
        MoveToUp(puzzle, x);
        MoveToDown(puzzle, x);
}

public void MoveToRight(int[] p, int i)
{
        if (i % col < col - 1)
        {
                int[] pc = new int[9];
                CopyPuzzle(pc, p);

                int temp = pc[i + 1];
                pc[i + 1] = pc[i];
                pc[i] = temp;

                Node child = new Node(pc);
                children.Add(child);
                child.parent = this;
        }
}

public void MoveToLeft(int[] p, int i)
{
        if (i % col > 0)
        {
                int[] pc = new int[9];
                CopyPuzzle(pc, p);

                int temp = pc[i - 1];
                pc[i - 1] = pc[i];
                pc[i] = temp;

                Node child = new Node(pc);
                children.Add(child);
                child.parent = this;
        }
}

public void MoveToUp(int[] p, int i)
{
        if (i - col >= 0)
        {
                int[] pc = new int[9];
                CopyPuzzle(pc, p);

                int temp = pc[i - 3];
                pc[i - 3] = pc[i];
                pc[i] = temp;

                Node child = new Node(pc);
                children.Add(child);
                child.parent = this;
        }
}

public void MoveToDown(int[] p, int i)
{
        if (i + col < puzzle.Length)
        {
                int[] pc = new int[9];
                CopyPuzzle(pc, p);

                int temp = pc[i + 3];
                pc[i + 3] = pc[i];
                pc[i] = temp;

                Node child = new Node(pc);
                children.Add(child);
                child.parent = this;
        }
}

public void PrintPuzzle()
{
        Console.WriteLine();
        int m = 0;
        for (int i = 0; i < col; i++)
        {
                for (int j = 0; j < col; j++)
                {
                        Console.Write(puzzle[m] + " ");
                        m++;
                }
                Console.WriteLine();
        }
}

public bool IsSamePuzzle(int[] p)
{
        bool samePuzzle = true;
        for (int i = 0; i < p.Length; i++)
        {
                if (puzzle[i] != p[i])
                {
                        samePuzzle = false;
                }
        }

        return samePuzzle;
}

public void CopyPuzzle(int[] a, int[] b)
{
        for (int i = 0; i < b.Length; i++)
        {
                a[i] = b[i];
        }
}
```

```
        public bool GoalTest()
        {
            bool isGoal = true;
            int m = puzzle[0];

            for (int i = 0; i < puzzle.Length; i++)
            {
                if (m > puzzle[i])
                {
                    isGoal = false;
                }
                m = puzzle[i];
            }

            return isGoal;
        }
    }
}
```

BFS.cs

```
namespace Lab5
{
    class BFS
    {
        public BFS()
        {

        }

        public List<Node> BreadthFirstSearch(Node root)
        {
            List<Node> PathToSolution = new List<Node>();
            List<Node> OpenList = new List<Node>();
            List<Node> ClosedList = new List<Node>();

            OpenList.Add(root);
            bool goalFound = false;

            while (OpenList.Count > 0 && !goalFound)
            {
                Node currentNode = OpenList[0];
                ClosedList.Add(currentNode);
                OpenList.RemoveAt(0);

                currentNode.ExpandMove();

                for (int i = 0; i < currentNode.children.Count; i++)
                {
                    Node currentChild = currentNode.children[i];
                    if (currentChild.GoalTest())
                    {
                        Console.WriteLine("Goal Found");
                        goalFound = true;

                        PathTrace(PathToSolution, currentChild);
                    }

                    if (!Contains(OpenList,currentChild) &&
!Contains(ClosedList,currentChild))
                    {
                        OpenList.Add(currentChild);
                    }
                }
            }

            return PathToSolution;
        }

        public void PathTrace(List<Node> path, Node n)
        {
            Console.WriteLine("Tracing Path...");
            Node current = n;
            path.Add(current);

            while (current.parent != null)
            {
                current = current.parent;
                path.Add(current);
            }
        }

        public static bool Contains(List<Node> list, Node c)
        {
            bool contains = false;

            for (int i = 0; i < list.Count; i++)
            {
                if (list[i].IsSamePuzzle(c.puzzle))
                {
                    contains = true;
                }
            }

            return contains;
        }

    }
}
```

Program.cs

```
namespace Lab5
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] puzzle =
                {
                    1,2,3,
                    5,6,0,
                    7,8,4
                };

            Node root = new Node(puzzle);
            BFS ui = new BFS();

            List<Node> solution = ui.BreadthFirstSearch(root);
            if (solution.Count > 0)
            {
                for (int i = 0; i < solution.Count; i++)
                {
                    solution[i].PrintPuzzle();
                }
            }
            else
            {
                Console.WriteLine("No path to solution is found");
            }
            Console.Read();
        }
    }
}
```